Kapitola: Primitívna rekurzia (pokrač.)

1.1 Predikáty a ďalšie funkcie

„Predikát“ je matematický pojem označujúci funkciu, ktorá vracia boolovskú hodnotu. V našom formalizme primitívnej rekurzie môžeme jednoducho za predikáty považovať funkcie, ktoré vracajú len hodnoty 0 (false) a 1 (true).

1.1.1 Signum a negácia

Inou možnosťou, ako definovať predikáty, by bolo interpretovať celé čísla ako pravdivostné hodnoty v štýle jazyka C: 0 je false, čokoľvek kladné je true. Oba spôsoby sú samozrejme ekvivalentné čo do vyjadrovacej sily. Formálne si to vieme zdôvodniť tak, že ukážeme primitívnu rekurzívnosť funkcie signum:

$$sgn(x) = \begin{cases} 0 & \leftarrow x = 0 \\ 1 & \leftarrow \text{inak} \end{cases}$$

Signum vieme zostrojiť primitívnou rekurziou z funkcie z a funkcie j_2 (binárnej konštanty 1): $sgn \equiv PR[z, j_2]$.

Ešte sa nám bude hodiť funkcia $\overline{sgn}$ určená predpisom $\overline{sgn}(x) = 1 - sgn(x)$. Tá z nuly vyrobí jednotku a z nenuly nulu, budeme ju teda pri predikátoch používať ako negáciu. Vyrobiť $\overline{sgn}$ si môžeme buď podobne ako sgn, teda primitívnou rekurziou, alebo kompozíciou unárnej jednotky, sgn a odčítania: $\overline{sgn} \equiv Comp[sub, j_1, sgn]$.

1.1.2 Porovnania a logické spojky

Testy na rovnosť a nerovnosť vieme ľahko spraviť pomocou nášho odčítania. Napr. test, či $x > y$, vieme spraviť tak, že sa pozrieme, či je hodnota $sub(x, y)$ kladná. Teda $gt \equiv Comp[sgn, sub]$ a analogicky $leq \equiv Comp[\overline{sgn}, sub]$. Na porovnanie opačným smerom potrebujeme odčítanie s opačným poradím argumentov, teda použijeme pomocnú funkciu m z definície sub: bude $lt \equiv Comp[sgn, m]$ a analogicky $geq \equiv Comp[\overline{sgn}, m]$.

Ak už máme primitívne rekurzívne predikáty p a q, ľahko zostrojíme predikáty pre ich logický and a logický or: na logický and stačí použiť súčin, na logický or súčet ich hodnôt. Presnejšie, $Comp[mul, p, q]$ je nový predikát, ktorý je pravdivý len vtedy, keď je pravdivé aj p, aj q. A podobne, $Comp[sgn, Comp[add, p, q]]$ je predikát pravdivý len vtedy, keď je pravdivý aspoň jeden z p a q. (Všimnite si technický detail: pri sčítaní sme ešte pridali kompozíciu so sgn, aby sme nevracali hodnotu 2, ak sú pravdivé oba predikáty.)

1.2 Meta-veta o if-e

Ako ďalší dobrý nástroj na výrobu nových primitívne rekurzívnych funkcií si teraz môžeme vysloviť a dokázať nasledovnú vetu o if-e:

Nech $f_1, \ldots, f_{k+1}$ a $g_1, \ldots, g_k$ sú primitívne rekurzívne funkcie s rovnakou aritou m. Položme $g_{k+1} \equiv 1$ a definujme funkciu h nasledovne: $h(\overline{x})$ je rovné $f_i(\overline{x})$, kde i je najmenšie prirodzené číslo, pre ktoré $g_i(\overline{x}) > 0$. Potom h je tiež primitívne rekurzívna.

Myšlienka dôkazu: $sgn(g_1(\overline{x})) \cdot f_1(\overline{x})$ je buď $f_1(\overline{x})$ alebo 0, podľa toho, či $g_1(\overline{x}) > 0$. Takto vyrobíme výrazy, ktoré majú hodnotu buď 0 alebo f_i, podľa toho, či nastala vhodná kombinácia pravdivostných hodnôt. No a na záver zostrojíme h tak, že nasčítavame všetky tieto výrazy.

1.3 Číslovacie funkcie a veta o p. r. časovej zložitosti

Čo už vieme: Vo formalizme primitívnej rekurzie vieme definovať práve tie programy, ktoré vieme počítať programami s jednoduchými for-cyklami. (Teda programy bez while-cyklov, rekurzie a podobných zveriniek – o týchto zatiaľ netušíme, či vôbec sú primitívne rekurzívne.)

Otvorené otázky: Existujú nejaké ďalšie veci ekvivalentné primitívnej rekurzii? Iné pohľady, cez ktoré vieme povedať čo je a čo nie je primitívne rekurzívne? Sú vlastne na niečo dobré while cykly a všeobecná rekurzia? Sure, umožnia nám napísať program, ktorý pre niektoré vstupy neskončí, ale je to všetko? Aký je súvis medzi primitívnou rekurziou a napr. tým, čo poznáme z Foje ako rekurzívne jazyky? Ak nám niekto zaručí, že program vždy zastaví, musí nutne počítať primitívne rekurzívnu funkciu?

Našim najbližším cieľom bude pomocou primitívne rekurzívnych funkcií vedieť zakódovať dvojicu prirodzených čísel do jedného. Chceme teda také primitívne rekurzívne funkcie $c(x, y)$, $l(x)$ a $r(x)$, aby platilo:

- c je prostá
- pre každé x, y je $l(c(x, y)) = x$
- pre každé x, y je $r(c(x, y)) = y$

Ako na to? Zoraďme si všetky dvojice prirodzených čísel do postupnosti. Systematicky to môžeme spraviť napr. tak, že ich zoradíme primárne podľa súčtu a sekundárne podľa prvého z nich:

$$(0,0),\ (0,1),\ (1,0),\ (0,2),\ (1,1),\ (2,0),\ (0,3),\ (1,2),\ \ldots$$

Teraz $c(x, y)$ definujeme ako zero-based index dvojice (x, y) v tomto poradí.

Zjavne $c(x, y) = (1 + \cdots + (x + y)) + x = \frac{(x+y)(x+y+1)}{2} + x$, preto c je primitívne rekurzívna.

Použijeme veľký arzenál na dôkaz, že aj l je primitívne rekurzívna: Určite $\forall x : l(x) \leq x$ a tiež $\forall x : r(x) \leq x$. Preto stačí vyskúšať všetky (a, b) z rozsahu od $(0, 0)$ po (x, x) a nájsť tú dvojicu, ktorej kód je x.

Python:

```
def l(x):
  for a in range(0,x+1):
    for b in range(0,x+1):
      if c(a,b)==x:
        return a
```

Formálne to isté ide napr. cez dve vnorené primitívne rekurzie (koniec koncov, primitívna rekurzia je vlastne jednoduchý for-cyklus).

Príklad: Fibonacciho postupnosť je primitívne rekurzívna. Jeden možný dôkaz: spravíme si pomocnú funkciu $helper(n)$, ktorá bude vracať kód dvojice (F_n, F_{n+1}).

Python:

```
def helper(n):
  if n==0:
    return (0,1)
  else:
    x,y = helper(n-1)
    return (y,x+y)
```

Intuitívna rekurzívna definícia:

$$\begin{aligned} helper(0) &= c(0,1) \\ helper(n+1) &= c(\ r(helper(n)),\ l(helper(n)) + r(helper(n))\ ) \end{aligned}$$

$$fib(x) = l(helper(x))$$

A ešte pre názornosť úplne formálne (až na vynechanie definícií c, l a r):

$$\begin{aligned} add &\equiv PR[P_1^1, Comp[s, P_1^3]] \\ j &\equiv Comp[s, z] \\ addpair &\equiv Comp[add, l, r] \\ almost &\equiv Comp[c, r, addpair] \\ g &\equiv Comp[almost, P_1^2] \\ helper &\equiv PR[j, g] \\ fib &\equiv Comp[l, helper] \end{aligned}$$

Takto vieme kódovať do celého čísla ľubovoľnú konštantne veľkú n-ticu. Dá sa však ísť ešte ďalej. Poučení Leibnizom môžeme ľubovoľnú konečnú postupnosť ľubovoľnej dĺžky zakódovať do mocnín prvočísel. Napr. takto: Kódom $(a_1, \ldots, a_k)$ bude

$$p_1^{a_1} \cdot \ldots \cdot p_k^{a_k} \cdot p_{k+1}$$

kde p_i je i-te prvočíslo. (Všimnite si, že na koniec sme pridali p_{k+1} ako zarážku, aby sme vedeli rozlíšiť napr. medzi kódom postupnosti $(4, 7)$ a postupnosti $(4, 7, 0, 0)$.)

Funkciu $div(x, y)$ vieme definovať pre $y > 0$ nasledovne:

$$div(x, y) = \sum_{1 \leq i \leq x} leq(iy, x)$$

Z nej ľahko definujeme *mod*, počet deliteľov, test prvočíselnosti, funkciu vracajúcu n-té prvočíslo a pomocou nich tiež funkcie na prácu s takto kódovanými postupnosťami.

Vieme reprezentovať pásku DTS ako kód konečnej postupnosti znakov, konfiguráciu DTS ako usporiadanú trojicu čísel (číslo stavu, pozícia hlavy, číslo kódujúce obsah pásky). Pre konkrétnu δ-funkciu DTS vieme definovať funkciu ktorá pre vstup=konfiguráciu vráti výstup=nasledujúcu konfiguráciu.

Veta o primitívne rekurzívnej časovej zložitosti: Nech existuje DTS A počítajúci funkciu $f(x)$, pričom existuje primitívne rekurzívna funkcia t taká, že $\forall n$: A na vstupe n spraví nanajvýš $t(n)$ krokov. Potom f je primitívne rekurzívna.

Dôsledok: Ak vôbec existujú totálne Turingovsky vypočítateľné funkcie, ktoré nie sú primitívne rekurzívne, musí ísť o funkcie, ktoré sa počítajú neuveriteľne ťažko.